MORRIS & AMATONG, P.C.
INTELLECTUAL PROPERTY LAW INCLUDING PATENT, TRADEMARK, COPYRIGHT LAW, UNFAIR COMPETITION AND RELATED MATTERS
10260 WESTHEIMER, SUITE 360
HOUSTON, TEXAS 77042

PAULA D. MORRIS
ALBERTO Q. AMATONG, JR.

RECEIVED
CENTRAL FAX CENTER
MAR 12 2007

PART 2 OF 2 * **Continuation of Foreign Patent Reference pgs. 18- Search Report. Total of 8 pages**

Certificate of Facsimile Transmission Under 37 CFR 1.8

I hereby certify that this correspondence is being transmitted via facsimile transmission in accordance with 37 CFR 1.8 on the date indicated below addressed to:

Commissioner for Patents
PO BOX 1450
Alexandria, VA 22313-1450

on

March 12, 2007	(571) 273-8300	* **No. of Pages including this cover sheet:** 36 *
Date	**Facsimile Number**	

[illegible]
Signature

Linda Sengvong
Typed or Printed Name

Attached are the following pages:

Part 1 of 2:
1. Response to Office Action Mailed October 11, 2006 [7 pages];
2. Petition for 2-Month Extension of Time [1 page].

Part 2 of 2:
* 3. Foreign Patent Reference GB 2,363,215 [35 pages]; and

Applicant:	Guidry	**Group Art Unit:**	2128
Serial No.:	10/710,823	**Examiner:**	Lo, Suzanne
Fling Date:	August 5, 2004	**Atty. Docket No.:**	19.0355
Title:	Dynamic Generation of Vector Graphics and Animation of Bottom Hole Assembly		

The Patent Office

PATENTS • DESIGNS • TRADE MARKS • COPYRIGHT •

INVESTOR IN PEOPLE

Application No:	GB 9928340.0	**Examiner:**	Nigel Hanley
Claims searched:	1-19	**Date of search:**	1 October 2001

Patents Act 1977
Search Report under Section 17

Databases searched:

UK Patent Office collections, including GB, EP, WO & US patent specifications, in:

UK Cl (Ed.S): G4A(APL);

Int Cl (Ed.7): G06F 9/44, 9/45

Other: ONLINE: WPI, EPODOC, JAPIO, INSPEC, TDB, ELSEVIER

Documents considered to be relevant:

Category	Identity of document and relevant passage		Relevant to claims
A,E	WO 00/22519 A1	ALCATEL - See whole document. Note the parsing of each line of assembly code into a data structure such that each line is accessible as individual fields such as as opcodes. Each opcode is then examined by traversal of decision tree to identify the assembler instruction and translation into source code.	
A	EP 0933699 A2	NEC - See whole document. Note the use of the symbol table to resolve assembly symbol definitions into language statements in "C"	
A	US 5946484 A	SOURCE - See whole document. Note example of using pattern matching to determine source code.	
X	US 5881290 A	ALLEN-BRADLEY - See whole document especially Fig 5 and Column 8 Line 34-Column 9 line 16. Note the use of an instruction table and symbol table to provide additional information for generating the source code.	1-5,9-11, 13

X	Document indicating lack of novelty or inventive step	A	Document indicating technological background and/or state of the art.
Y	Document indicating lack of inventive step if combined with one or more other documents of same category.	P	Document published on or after the declared priority date but before the filing date of this invention.
		B	Patent document published on or after, but with priority date earlier than, the filing date of this application.
&	Member of the same patent family		

An Executive Agency of the Department of Trade and Industry expression calculator processes the relocations having an offset 0000 shown in Figure 7a. Thus, when the all the relocations at offset 0000 have been processed the top of the expression stack 34 contains the original expression. This is then retrieved by the expression calculator 32 and supplied to the disassembler 22 together with the first instruction 10000C8 . The resulting display is shown at the bottom of Figure 7a consisting of the program count A, the hexadecimal assembled object code B, the disassembled instruction in source code corresponding to the hexadecimal object code (and including the final value 0 in the first instruction) and the expressions, d, which is used to calculate the value of 0 in the first instruction, derived from the relocation data by the expression calculator and expression stack.

The example of Figure 7a similarly illustrates the same sequence with respect to the second instruction at program count 0004.

Figure 8 illustrates how the lister operates to deal with the second example discussed above. An object code module 3 contains the code sequence discussed above in relation to example 2. In line with Figure 5, the object code module includes a plurality of alternative code sequences, denoted O1,O2 in Figures 5 and 8. In this particular example, each code sequence is a single line of code, although it will readily be appreciated that a plurality of lines of code could be provided for each alternative code sequence. The relocation section is shown to include three conditional relocations R_IF, R_ELSE and R_ENDIF attached to the appropriate offsets for the optional code lines in the section data. When the first optional code line O1 at program count 0008 is read by the data reader 11, the relocations reader determines that a conditional relocation R_IF is present at the same offset, and supplies the relocation to the directive processor 30. The directive processor uses this information to regenerate the conditional assembler directive which gave rise to the conditional relocation. The conditional assembler directive and the instruction at program count 0008 are supplied to the disassembler. Because conditional assembler directives are not executable instructions themselves (that is they do not themselves form part of the executable program), when the source code containing the conditional assembler directives is assembled, the program counter is not incremented. Thus, the conditional assembler directive is displayed with a program count corresponding to that of the original optional code line. The same procedure is carried out for the optional code line O2 at program count 0000C and also for the ending code line at program count 0010 which is likewise found to have a final conditional relocation R_ENDIF at that offset.

Figure 9 illustrates how the lister operates to deal with events generated in response to event relocations. Each instruction within a data section of the executable program may have one or more events associates with it, denoted by the symbol Ø where Ø can take any value of integer from 1 to n. In the example illustrated in Figure 9 the instruction BARØ at program count 0000 is associated with the event "WARNING((((FOO-$)-4)&1)==Ø)", where $ represents the program count. Initially the operation of the lister is the same as that described in relation to Figures 7a and 7b. The relocation reader 16 supplies the expression relocations R_PUSH to R_EQ having the same offset as the instruction to the expression calculator 32 which, using the expression stack 34 regenerates the initial expression. However when the event relocation R_ASSERT WARNING is received by the relocation reader 16 it is supplied to the event calculator 36. The event calculator retrieves the expression from the top of the expression stack 34, generates the event "WARNING((((FOO-$)-4)&1)==Ø)", and places this on top of the event stack 38. If any further events are associated with the same instruction these are then processed in turn and also placed on the event stack 38 so that when all the events have been processed the event stack 38 will hold Ø events. When it is detected that there are no more events associated with an instruction (determined by the relocation offset changing value), the events are popped from the event stack 38 and supplied to the disassembler 22 together with the original instruction 0000a000.

In the above description it has been assumed that the program count in the object code module is the same as the offset identified in the respective relocation. In fact however it will readily be appreciated that the offset identified in the relocation can be taken from an index value representing the base of the object code module or the base of the particular section. Thus, the offset identified in the relocation merely allows the correct location in the section data to be identified and need not be identical to the program count itself.